2

NAVAL POSTGRADUATE SCHOOL
Monterey, California

THESIS

A PERFORMANCE STUDY OF THE CONCURRENCY CONTROL ALGORITHMS IN HIERARCHICAL NETWORK WITH PARTITIONED DATABASE

by

Shin, Eon Seok

March 1990

Thesis Advisor: Chyan Yang

Approved for public release;distribution is unlimited

Unclassified

security classification of this page

REPORT DOCUMENTATION PAGE

1a Report Security Classification Unclassified	1b Restrictive Markings
2a Security Classification Authority	3 Distribution Availability of Report
2b Declassification Downgrading Schedule	Approved for public release: distribution is unlimited.
4 Performing Organization Report Number(s)	5 Monitoring Organization Report Number(s)
6a Name of Performing Organization Naval Postgraduate School; 6b Office Symbol *(if applicable)* 52	7a Name of Monitoring Organization Naval Postgraduate School
6c Address *(city, state, and ZIP code)* Monterey, CA 93943-5000	7b Address *(city, state, and ZIP code)* Monterey, CA 93943-5000
8a Name of Funding Sponsoring Organization; 8b Office Symbol *(if applicable)*	9 Procurement Instrument Identification Number
8c Address *(city, state, and ZIP code)*	10 Source of Funding Numbers: Program Element No; Project No; Task No; Work Unit Accession No

11 Title *(include security classification)* A PERFORMANCE STUDY OF THE CONCURRENCY CONTROL ALGORITHMS IN HIERARCHICAL NETWORK WITH PARTITIONED DATABASE

12 Personal Author(s) Shin, Yon Seok

13a Type of Report	13b Time Covered	14 Date of Report *(year, month, day)*	15 Page Count
Master's Thesis	From To	March 1990	49

16 Supplementary Notation The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.

17 Cosati Codes			18 Subject Terms *(continue on reverse if necessary and identify by block number)*
Field	Group	Subgroup	Concurrency Control, Hierarchical Computer Network

19 Abstract *(continue on reverse if necessary and identify by block number)*

It is common to have a hierarchical communication network in a military environment. If we consider each node in the network as a computer site then we have a hierarchical computer network. In a hierarchical computer network, because the need of resource sharing we now have a distributed processing system. In this system a parent node may have duplicate records of all its children. Any update of record in a data file has to be reflected in other nodes that keep the duplicates. We need a concurrency control mechanisms to gaurantee the integrity of the distributed database and the serializability of concurrent updates.

This thesis is the first to investigate the performance in hierarchical networks of two widely cited concurrency control mechanisms: locking based and timestamped. Various parameters are investigated in this research: number of nodes, level of network, transaction arrival rates, and message transmission speeds, etc. We present the problem, explain the algorithms used in our simulation, analyze the results, and discuss the findings.

20 Distribution Availability of Abstract ☒ unclassified unlimited ☐ same as report ☐ DTIC users	21 Abstract Security Classification Unclassified
22a Name of Responsible Individual Chyan Yang	22b Telephone *(include Area code)* (408) 646-2266; 22c Office Symbol

DD FORM 1473,84 MAR

83 APR edition may be used until exhausted
All other editions are obsolete

security classification of this page

Unclassified

Approved for public release; distribution is unlimited.

A Performance Study of the Concurrency Control Algorithms
in Hierarchical Network with Partitioned Database

by

Shin, Eon Seok
Captain, Korean Army
B.S., Korea Military Academy, 1983

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN ELECTRICAL ENGINEERING

from the

NAVAL POSTGRADUATE SCHOOL
March 1990

Author: Shin Eon Seok

Shin, Eon Seok

Approved by: Chyan Yang

Chyan Yang, Thesis Advisor

Myung W. Suh

Myung W. Suh, Second Reader

John P. Powers

John P. Powers, Chairman,
Department of Electrical and Computer Engineering

ABSTRACT

It is common to have a hierarchical communication network in a military environment. If we consider each node in the network as a computer site then we have a hierarchical computer network. In a hierarchical computer network, because the need of resource sharing, we now have a distributed processing system. In this system a parent node may have duplicate records of all its children. Any update of a record has to be reflected in other nodes that keep the duplicates. We need a concurrency control mechanism to guarantee the integrity of the distributed database and the serializability of concurrent updates.

This ~~paper~~ is the first to investigate the performance in hierarchical networks of two widely-cited concurrency control mechanisms, locking based and timestamp. Various parameters are investigated in our research: number of nodes, level of network, transaction arrival rates, and message transmission speeds, etc. We present the problem, explain the algorithms used in our simulation, analyze the results, and discuss the findings.

TABLE OF CONTENTS

ACKNOWLEGEMENTS

I wish to express my sincere appreciation to my thesis advisor, Professor Chyan Yang. He was a consistent wellspring of sound advice, technical competence, professional assistance, and moral support. Without his assistance, this thesis could not have been undertaken.

I am also thankful to Professor Myung W. Suh who gave me many ideas and encouraged me to finish this thesis.

I would lile to thank to my friend Ryoo, Moo Bong who taught me the GML with excellent knowledge and warm heart.

I would also like to thank to my wife, Mi Kyeong, and my dauther, Haenuli, for their support and patience away from home during the life in the United States.

Now it is time to go home and I'd like to thank my family, far away in Korea who have stayed in my heart all the way through.

LIST OF FIGURES

I. INTRODUCTION

A. BACKGROUND

In the course of the development of computer and communication technology several important technical achievements have been introduced. One of the most significant techniques among these is distributed data processing. A distributed system consists of a set of computers located in different sites connected by a communication network. Programs are running on each of these computers and the programs access local or remote resources, such as a database.

The major advantage of a distributed system is to provide low-cost availability of resources of the system by localizing accesses and providing insulation against failures of individual components. Since many users can be concurrently accessing the system, it is essential that a distributed system should provide a high degree of concurrency. Due to the requirement of data sharing, the distributed system may be designed with more than one copy of the same data in different locations. This partitioned and replicated distributed database requires synchronization to control the concurrent multiple updates and maintain its consistency. This synchronization is called the concurrency control algorithm.

There are two types of concurrency control approaches, local concurrent control and global concurrency control. The local concurrent control mechanism is developed to guarantee that a step of a transaction is executed as a single atomic operation at an individual node. The global concurrency control mechanism which is also called the update algorithm, controls the transactions between two different nodes [Ref. 1]. Many algorithms have been proposed in this field [Ref. 1] [Ref. 2] [Ref. 3]. These algorithms are usually complex and hard to understand. However most of the work on concurrency

control has concentrated on the development of new algorithms, and not as much attention has been given to the performance evaluation of the algorithms. Also, most performance studies are done in general network structure [Ref. 1] [Ref. 2] [Ref. 4]. For military settings, the line of authority is hierarchical, therefore military systems call for hierarchical computer networks in a distributed system environment. System managers or designers usually like to investigate performance differences and sometimes like to dynamically adapt the system to the transaction pattern. Therefore, it is important to analyze performance differences between different concurrency control algorithms for hierarchical networks. This research is to report the performance analysis of such networks.

B. DISTRIBUTED DATABASE

A distributed database system is hosted by collection of geographically separated computers called sites. Each sites has a unique identifier, and the sites are interconnected by a communications network. Sites communicate with each other by means of messages sent over the network. Messages may be arbitrarily delayed in the network, but it is assumed that all messages are eventually delivered.

The database consists of an unique set of entities, such as records or files, that are uniquely named and that serve as indivisible units of access. An entity is realized as one or more data objects, each of which is uniquely identified by an [entity-name, site-identifier] pair.

Replication of an entity occurs if it is represented by more than one object. Replication can result in improved performance if the cost of storing and maintaining (updating) copies is less than the cost of access. Also replication can improve availability in the sense that, if one site becomes inaccessible, then users may be able to access the data at other site. In our model it is assumed that each entity is partially replicated which means the entities are not replicated at all nodes but replicated at some nodes.

C. CONSISTENCY CONSTRAINTS

We would like to keep a database consistent. However, due to updating activity, the consistency constraints must be temporally violated. Two levels of consistency can be defined for multiple copies of data. Strong consistency is defined as the condition of having all copies of the data updated at the same time. Strong consistency is very desirable becuse all copies of data have the same update status at any time, but this always entails a considerable delay in response time to process the update for all copies. This is because the update of all copies must be delayed to the time of the last update of system. Weak consistency is defined as the condition of having the various copies of data converge to the same update status over time, but at any instant of time some copies may lag others in the number of updates processed. Delays in general will be reduced and more efficient use of resources is possible. In a weak consistency system some copies of data will be more up-to-date than others [Ref. 5]. Most algorithms are developed under weak consistency constraints because of efficiency, and our simulation also follow weak consistency constraints.

D. GENERAL DEFINITIONS

We now give four important definitions that we need in the following discussions.

1. Hierarchical distributed data base system: A system with the database dispersed throughout a number of nodes, where each node can communicate in real time with one or more subordinate nodes, but with only one superior node and with sufficient processing power at each node to manipulate the database.
2. Concurrency control: The mechanism of maintaining logical data consistency in an environment of contention among multiple update sources.
3. Node: Complete computer system at a single location.
4. Transaction: Set of actions which transform a database from one consistent state into another consistent state [Ref. 5].

E. GENERAL ASSUMPTIONS

For our study we make the following assumptions:

1. The data bases are partially replicated in the system, which is more practical than completely replicated databases.
2. Although update transactions may add or delete database items, we consider the total number of data in database to be constant, that is, a static database.
3. All nodes have local concurrency control. We are not concerned with local concurrency control mechanism, but concentrate on the global concurrency control algorithms.
4. There is a communication system which allows any node to communicate with any other node.
5. No failures occur in the system, that is, the communication system never fails to deliver a message.
6. All transactions which are used for this study are update transactions.
7. There is no multiprocessing at each node. The transactions which arrive at certain nodes are served serially. Since most transactions are small, the schedule algorithm in each node should not have a noticeable effect on the response time and results obtained with the model.

II. CONCURRENCY CONTROL ALGORITHMS

Concurrency control is the activity of coordinating concurrent accesses to a database in a multi-user database management system. Concurrency control permits users to access a database in a multiprogrammed fashion while preserving the illusion that each user is executing alone on a dedicated system. The main technical difficulty in achieving this goal is deconfliction. Database update performed by one user must not interfere with database retrievals and updates performed by another. Although many algorithms have been proposed, most of them are a variation of two basic techniques, Two-Phase Locking and Time-Stamp Ordering. Alternatively we can interpret these two technique as centralized and distributed algorithms. The best concurrency control algorithm for a particular application depends on the system parameters. We will discuss these parameters in Chapter 3.

A. LOCKING-BASED CONCURRENCY CONTROL

Objects accessed by a transaction are locked in order to ensure their inaccessibility to other transactions while the database is temporarily in an inconsistent state. There are three lock-releated actions on objects:

1. (T, Lock_X, < A >) denotes a request by transaction T for an exclusive lock on the object < A >. The request is granted only when no other transaction is holding a lock of any type on object.
2. (T, Lock_S, < A >) denotes a request by transaction T for a shared lock on the object < A >. The request is granted only when no transaction is holding an exclusive lock on the object.
3. (T, Unlock < A >) denotes the release of any lock by transaction T on the object < A >.

A transaction is said to be well-formed if it reads an object only while it is holding a shared or exclusive lock on the object and if writes an object only while it is holding an exclusive lock on the object. Figure 1 illustrates well-formed transaction.

(T1, Lock_S, A)

(T1, Read_object, A)

(T1, Unlock, A)

(T1, Lock_S, B)

(T1, Read_object, B)

(T1, Unlock, B)

(T1, Lock_X, A)

(T1, Write_object, A)

(T1, Unlock, A)

(T1, Lock_X, B)

(T1, Write_object, B)

(T1, Unlock, B)

Figure 1. Well-formed Transaction

A transaction is Two-Phase if it does not issue a lock action after it has issued an unlock action. The first phase of the transaction, called the growing phase, begins with the first action and continues up to the first unlock action. The second phase of the transaction, called the shrinking phase, begins with the first unlock action and continues through the last action. A transaction is strong two-phase if all unlock actions are issued at the very end of the transaction. Figure 2 illustrates a typical Two-Phase locking action. The transaction T1 locks the data item A, and, after finishing all steps of the update, it unlocks the item A. T1 does not need additional lock action for item A.

We apply the Two-Phase Locking Algorithm to the centralized strategy which is known Centralized Locking Algorithm (CLA).

```
( T1, Lock_X, < A > )
( T1, Lock_X, < B > )
( T1, Read_object, < A > )
( T1, Write_object, < A > )
( T1, Read_object, < B > )
( T1, Write_object, < B > )
( T1, Unlock, < A > )
( T1, Unlock, < B > )
```

Figure 2. Two-Phase Locking Action

B. CENTRALIZED LOCKING ALGORITHM (CLA)

For the centralized locking algorithm, the lock manager is centralized at a single node, i.e., the central or root node. It manages the locks of all data elements of the distributed database. We will explain the steps of CLA.

1. An update transaction T arrives at node X.
2. Node X requests locks from the central node for all items referenced by transaction T.
3. The central node checks all the request locks. If all can be granted, then a granted message is sent back to the node X. If some items are already locked, then the request is queued.
4. When node X gets a "Grant" message for transaction T, the items requested by transaction T are read from the local database and updated values are computed.
5. A "Perform update T" message is sent to all other nodes, informing them of the update. Node X updates the values and stored in its local database.
6. When another node receives its "Perform update T"message, it performs the update.
7. When the central node receives its "Perform update T" message, it releases the locks of the involved items and then performs the update on the local database. Transactions that were waiting on the released locks are notified, and can continue their locking process at their central node.
   (End of CLA algorithm)

C. CONCURRENCY CONTROL ALGORITHMS BASED ON TIME-STAMP

Time-stamp ordering (T O) is a technique whereby a serialization order is selected a priori and transaction execution is forced to obey this order. Each transaction is assigned a unique time-stamp by the time-stamp manager (TM). There also is a time-stamp associated with each database item indicating the time when this item was updated. Time-stamp protocol is distributed in nature since all nodes which share the data item must participate in the decision making. So here we see the distributed voting algorithm using time-stamp which we simulated.

D. DISTRIBUTED VOTING ALGORITHM (DVA)

1. When transaction T arrives at a node X, it immediately reads the items and time-stamp of desired from the local database. We use $Ts(T,Di)_{node\ x}$ to denote this.
2. Transaction T visits the next node (i.e., node Y) and compare the time-stamp for the transaction base set items with the corresponding time-stamps in the local database. If any base set item is obsolete (i.e., $Ts(T,Di)_{node\ x} < Ts(Di)_{node\ y}$) then vote "Reject" and send the "Reject" message to originating node. The transaction will be restarted. If each base item is current (i.e., $Ts(T,Di)_{node\ x} > Ts(Di)_{node\ y}$) then vote "Ok".
3. After voting all the nodes which have the base item, if the transaction T got "Ok" at all nodes, then the time-stamp is assigned to the transaction (i.e., $Ts(T)$) and the "Accept" message with $Ts(T)$ is sent to all nodes.
4. If a node receives an "Accept" message and if $Ts(T) > Ts(T,Di)_{at\ each\ node}$ then the new values are stored in the local database and time-stamps of the base set items are changed. Otherwise, no modification is performed, since the values are obsolete.
(End of DVA)

III. SIMULATION MODEL

We built a simulator using SIMSCRIPT II.5 to examine the algorithms for our study. Each simulator has an update transaction generator that produces transactions. The items referenced by each transaction are selected randomly. The simulator then mimics the operation of the hierarchical system as it produces the transactions. Of course, the simulator does not read or write the data corresponding to a transaction; it only mimics this by requesting the necessary I O and CPU time from the servers. However, the simulator keeps track of such activities as granting locks, time-stamping the item values, and differed transactions. During the simulation, statistics such as average response time of transactions and the number of messages are collected.

Several model parameters are used:

1. Mean interarrival time of update at each node (It). We assume that interarrival time is Poisson distributed.
2. Mean base set size (Bs). We assume that the number of items referenced by an update transaction has discrete exponential distribution.
3. Number of items (M). This parameter describes the total number of data items in the system.
4. The number of nodes (N).
5. The network transmission time (T). In order to simplify the simulation, we assume that the time it take any message to go from any node to any other node is a constant T.
6. CPU time slice (Cs). The CPU time slice is the time it takes any CPU server to do a small computation; i.e., set lock.
7. CPU update compute time (Cu). Cu is the compute time required for updating one data item. In other words, the total time for one transaction is Cu multiplied by Bs.
8. I O time slice (Id). This is the time it takes to read or write a data item from the database and locks or time-stamps.

The following parameter values are common to most of the previous performance evaluation studies [Ref. 4] [Ref. 6]. It = 1 - 20 seconds, Bs = 2 - 10, N = 2 - 15 nodes,

T = 0.1 seconds M = 100 - 2000, Cs = 0.00001 seconds, Cu = 0.001 seconds, and Id = 0.025 seconds.

A. THE PERFORMANCE MEASURE

There are many variables one can choose to evaluate the performance of the system. In this study we will consider two most important variables:

1. UPDATE RESPONSE TIME. The response time of update is defined as the difference between the finish time and the time when the update arrived at the originating node. The average response time of update transactions will be investigated.
2. NUMBER OF MESSAGES. Another important performance variable is the number of messages that must be sent per update transaction. The local messages are considered internal to a node and are not counted here.

B. PARTITIONED DATABASE MODEL

In this section we explain the partitioned database model in a general computer network; modeling of the partitioned hierarchical database will be described in next section.

In a partitioned distributed database, which means a partially replicated database, items are not replicated at all nodes. As matter of fact, some items might not be duplicated at all. There might exist only a single copy of some items. From the point of view of a single node, it has a fraction or partition of the database. This partition can be identical to, completely disjointed from, or can overlap the partitions at other nodes.

In order to model the partitioned database, we choose the simple model (a this time it is not a hierarchical model) in which the data is replicated. As we mentioned before, the database is the fixed set of M shared named resources called items. Each item has a name and some value associated with it. For simplicity, we use the integer between 1 and M (i.e., $1 < i < M$).

1. DESCRIPTION PARAMETERS

Now we make some description parameters in order to describe the model. The set S(i) is the set of nodes which have a copy of the value of item i. The elements of S(i)

are the node identification numbers which have a copy of value of item i. We assume all sets S(i) are not empty. Another parameter is a fragment, which is defined as the set of items that have a same set S(i). We use the notation S(F) for the set of nodes where the F is stored. That is S(F) equals S(i) for all items i in F. These parameters are used for simulation. Figure 3 illustrates the very simple model and model description parameters for current control.

Let Tr1 be the transaction which arrives at node B and has the base set

Bs1 = [1, 2, 3], then

S(1) = [A, B, C, D]

S(2) = [A, B, C]

S(3) = [A, B, C, D]

For Tr1 F = [1, 3] and S(F) = [A, B, C, D].

Let Tr2 be the transaction which arrive at node D, and has the base set

Bs2 = [3, 5], then

S(3) = [A, B, C, D]

S(5) = [A, C, D, E]

For Tr2, F is null and S(F) is not defined.

2. DIRECTORY MANAGEMENT

Now we discuss the directory management. Suppose that a transaction arrives at a certain node with its base set. How does the transaction know what nodes have a copy of data item i? In the case of Tr1, which we used in the previous example, how does Tr1 or node B generate the set S(i)?

There are basically two kinds of solutions. The first method is that a complete directory of the whole database is replicated in each node. The other method is that the transaction broadcasts the message " Where is the item i located ? " to all nodes and waits for the response from other nodes which have the data item i [Ref. 1]. A directory is a mapping that produces the set S(i) or S(F), given the item name i or fragment F.

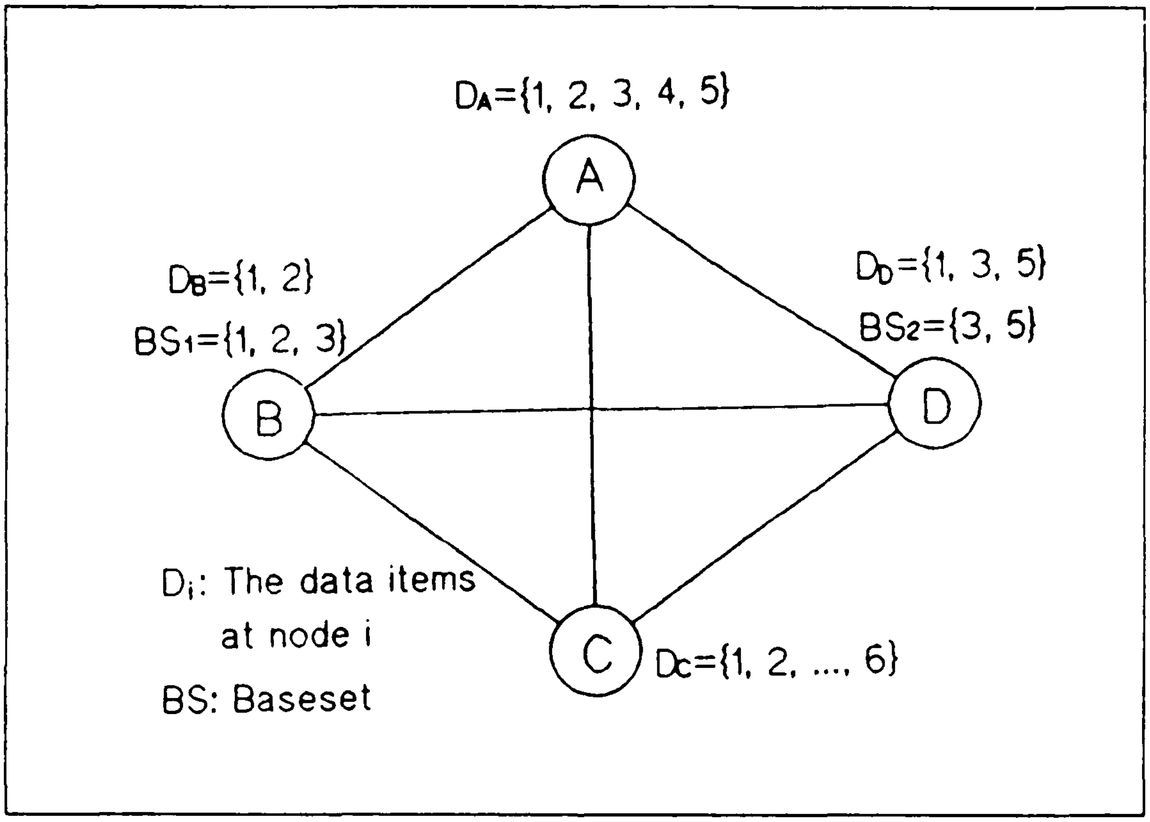

Figure 3. Simple Partitioned Database Model.

By consulting the directory, a transaction T will be able to find out the set S(i) corresponding to every item in Bs. In next chapter we take care of this problem in detail.

C. PARTITIONED HIERARCHICAL DATABASE MODEL

In the previous section we described the general model of a partitioned database. Now we will make a hierarchical model of the partitioned database. When we describe the hierarchical computer network, the level will be important factor. The level is the depth of the network from the root node, which is the central node of the network to the bottom nodes.

One of the characteristics of hierarchical distributed system is that each node has the authority to control a subordinate node. We make one definition here. The logical

control channel is the chain of control which can give authority for updating. This definition is very important because although there may be a physical connection between children nodes, it is not logical control channel.

Additional assumptions are adopted for the hierarchical model. Let D1 be the total set of data items in the database at node "1" and suppose node "1" has two subordinate nodes "2" and "3", and their total set of data items in their database are D2 and D3 then

D1 = D2 + D3 + data items of node 1's own.

We can generalize this. If node 1 has n children nodes, then

$$D_1 = \sum_{j=1}^{n} D_j + 1\text{'s own data items}$$

Where D_j are children node of D_1

So the root node which is the central node in hierarchical model has all the data items in the network. Another important assumption we use is that the transaction Tr can only request the update data items in its original node. If items which are referenced by a certain transaction are not located at that node, then the transaction is a wrong transaction. This is quite reasonable in a hierarchical distributed database model.

The directory of our model is somewhat like our database, so each node has its own directory and the root node has a whole directory of the system. Figure 4 illustrates the very simple model of a partitioned hierarchical database model. Let Tr1 arrive at node C with baseset Bs1=[1,2], then

S[1] = [A, B, C]

S[2] = [A, B, C]

and F = [1, 2], so S[F] = [A, B, C]

Let Tr2 arrive at node B with baseset Bs2 = [1, 5, 8], then

- S[1] = [A, B, C]
- S[5] = [A, B, D]

- S[8] = [A, B]
- and F is empty, therefore S[F] is not defined.

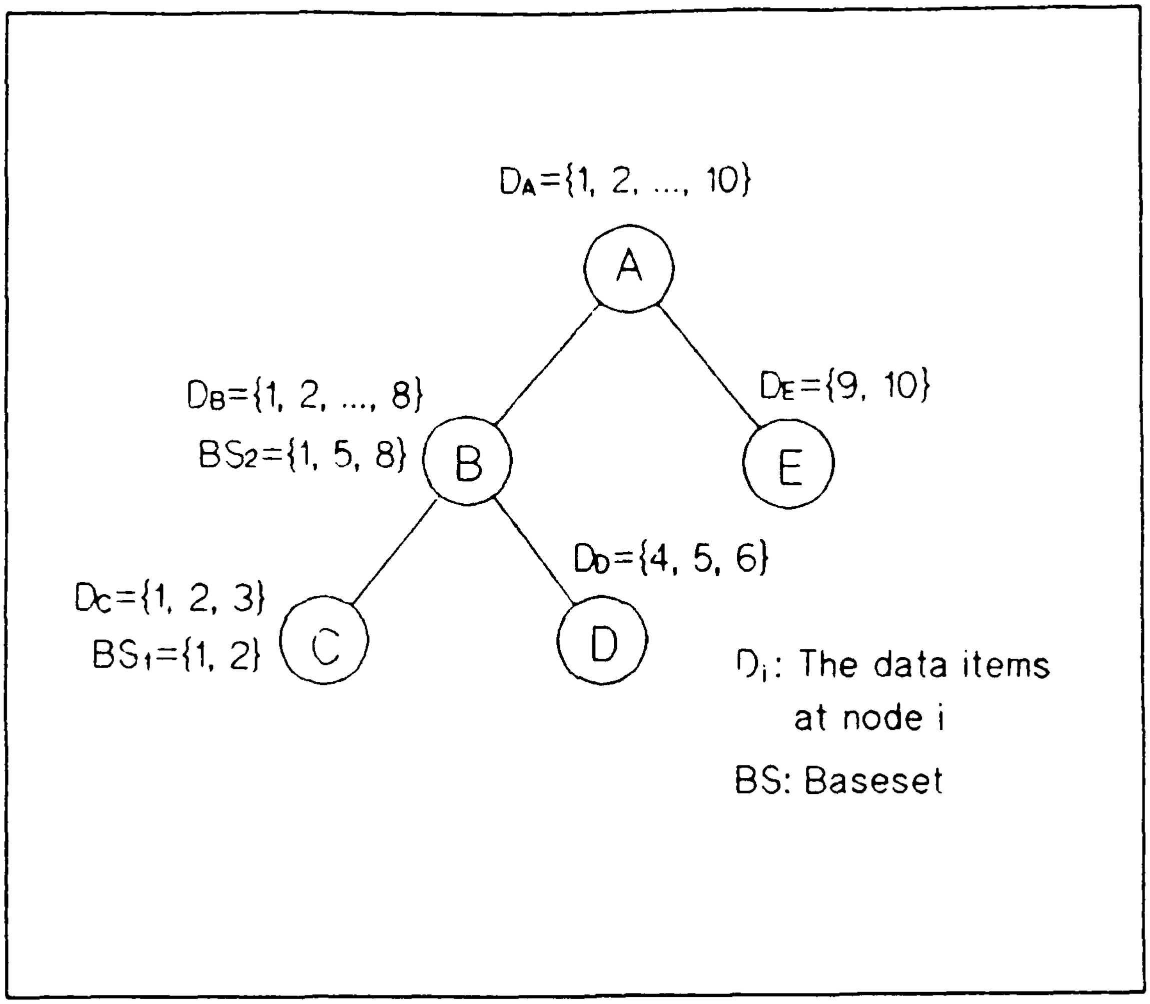

Figure 4. Partitioned Hierarchical Database Model

IV. ANALYSIS OF SIMULATION RESULTS

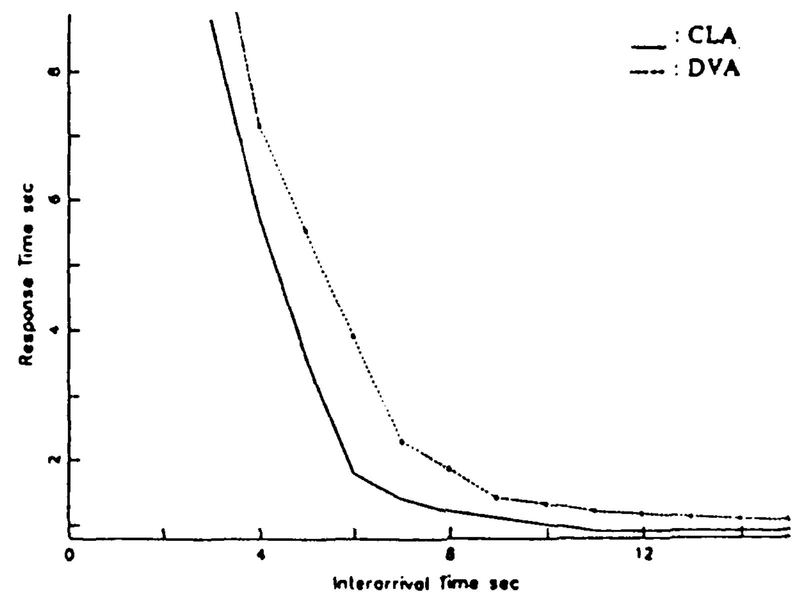

Figure 5. Effect of Interarrival Time on Response Time

We have tried numerous system settings and studied the sensitivity of each parameter separately. Important system parameters are the response time, the number of nodes in the network, the number of levels in the network, the interarrival rate of transactions, and the average system-wide traffic. All curves in this chapter are plotted by connecting discrete data points from simulation results.

Figure 5 shows the effect of the mean interarrival time on the mean response time. We notice that, the longer the interarrival time (low arrival rate), the shorter response time in both algorithms, but CLA outperforms the DVA.

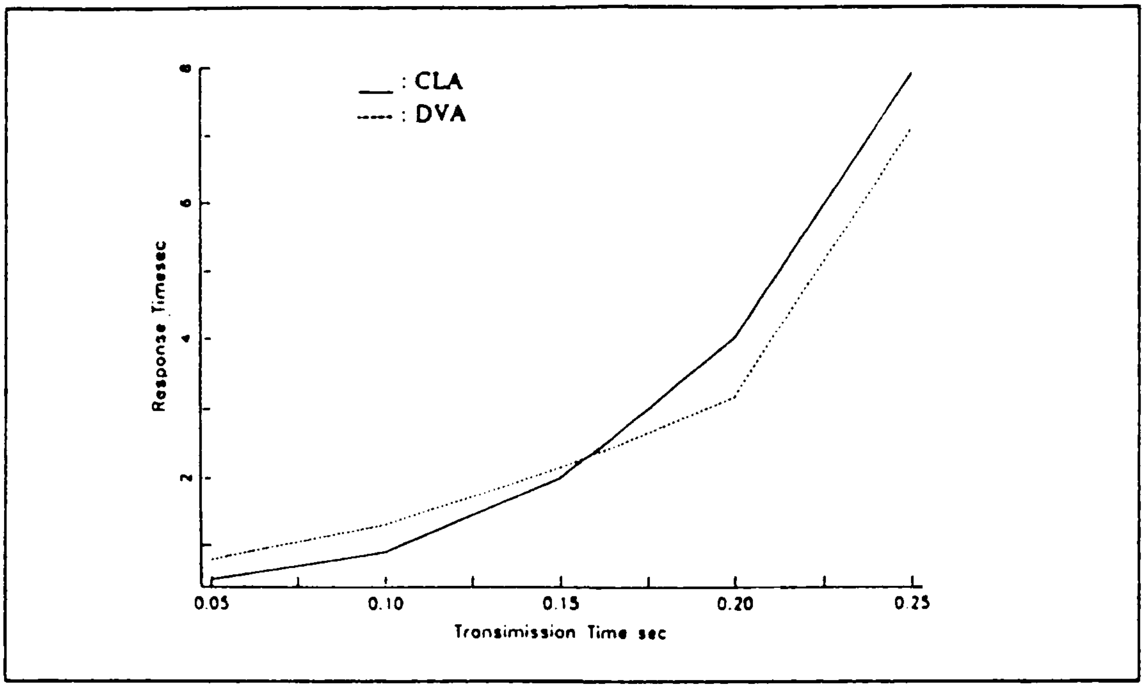

Figure 6. Effect of Transmission Time on Response Time

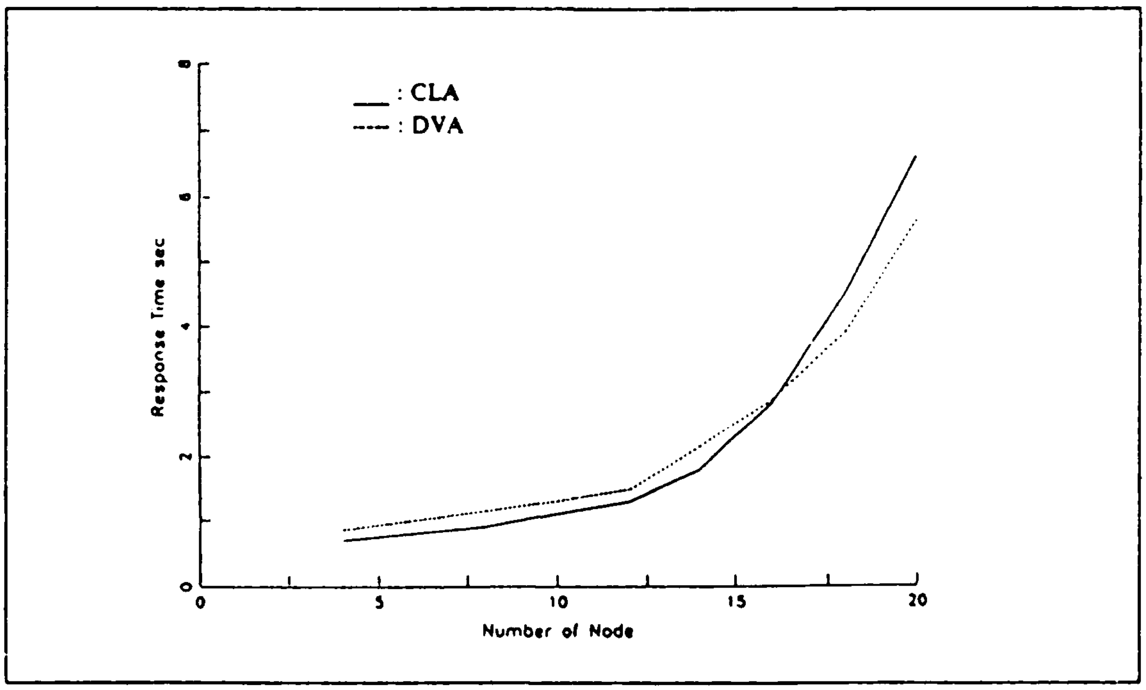

Figure 7. Effect of Number of Nodes on Response Time

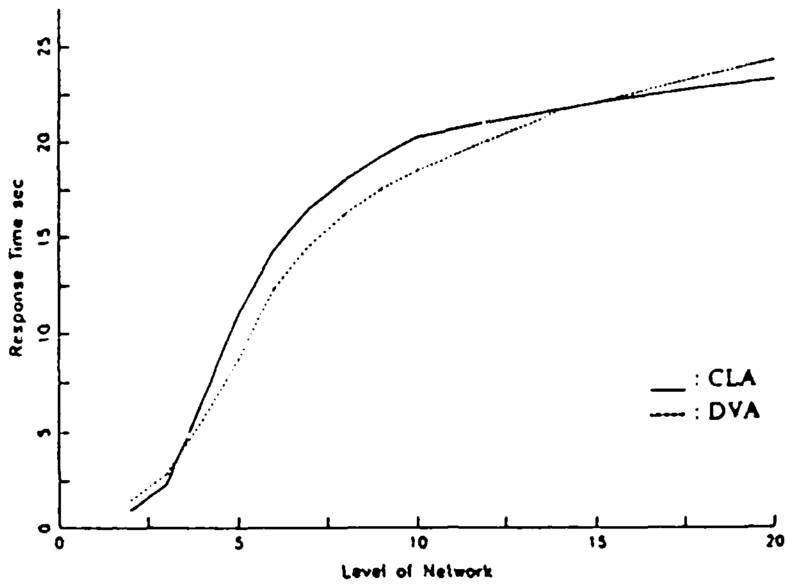

Figure 8. Effect of Level of Network on Response Time

This is because in CLA only the root node perform decision making. However depending on the level of the network, the results vary.(See Figure 7, Figure 9, Figure 10, and Figure 11).

The effect of transmission time on the response time of transaction is shown in Figure 6. Both algorithms exhibit the same behavior. The CLA appears more sensitive to the transmission time than the DVA. This is because the number of traffic in CLA is much higher than that of DVA (See Figures 12, and 13). Also the CLA begins to suffer a transmission delay when T reachs 0.16 sec. When transmission time is low, DVA is penalized by the coordination time among participants, while CLA is penalized for the time required to send the grant signal to the requesting node when the transmission time is above a certain threshold (0.16 in Figure 6).

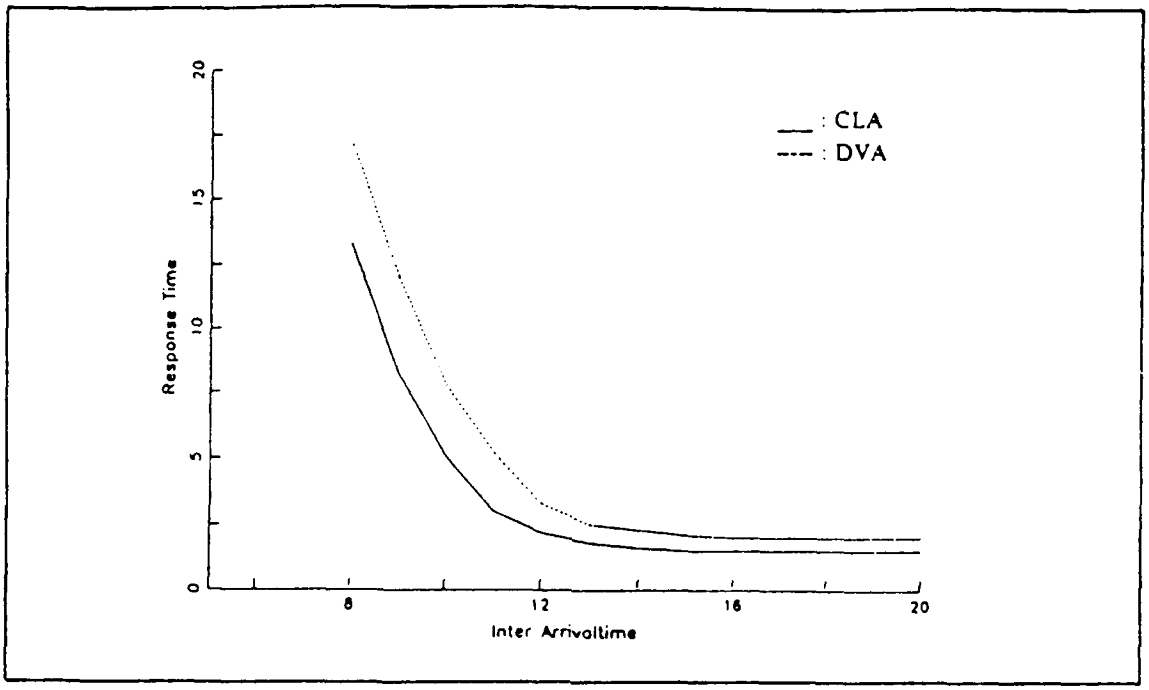

Figure 9. Effect of Interarrival Time on Response Time for Case I

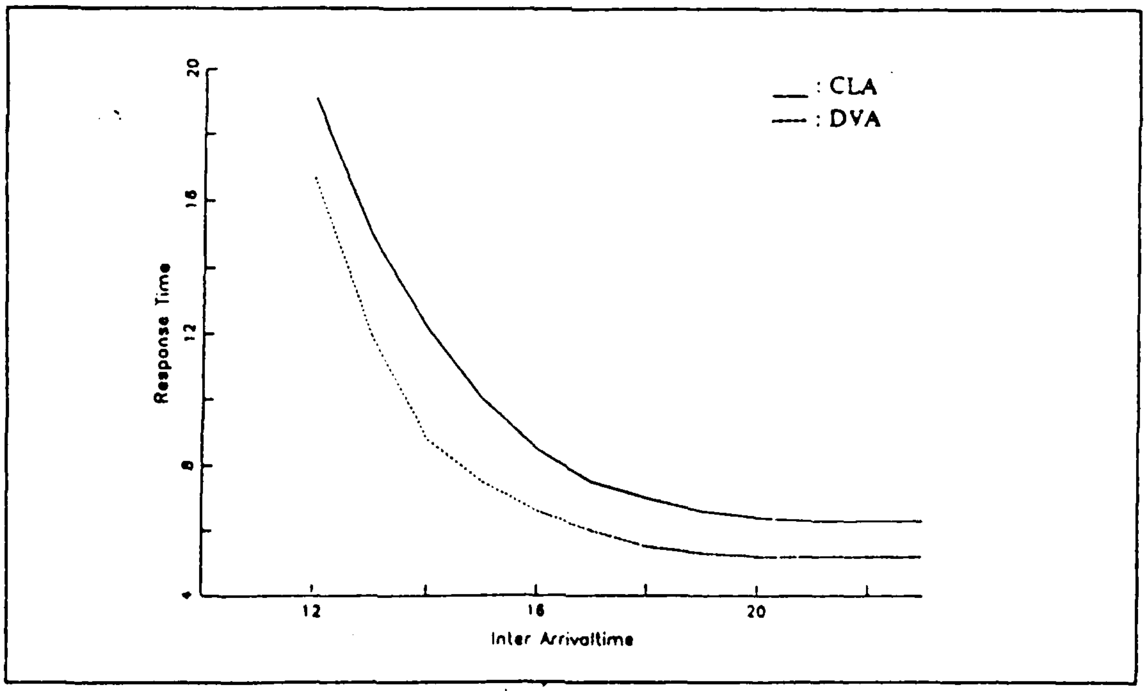

Figure 10. Effect of Interarrival Time on Response Time for Case II

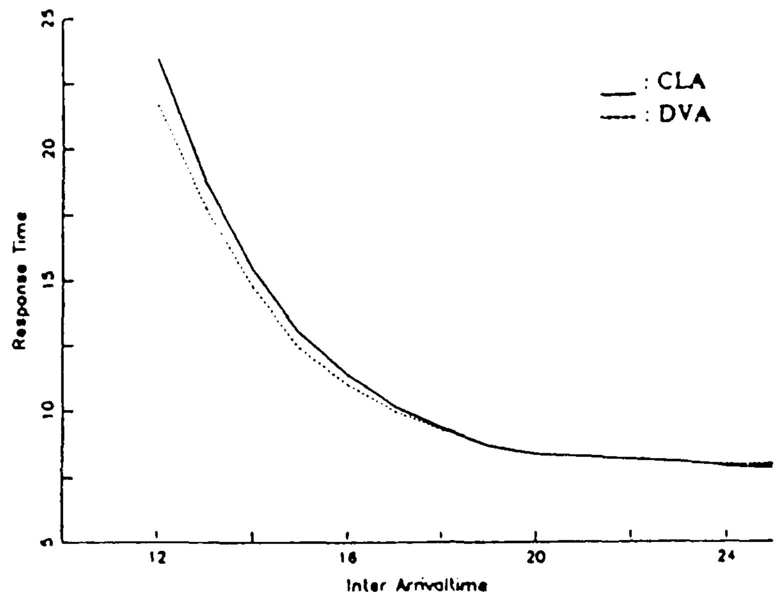

Figure 11. Effect of Interrival Time on Response Time for Case III

Figure 7 shows the effect of the number of nodes on response time. For both algorithms, increasing the number of nodes in the network increases the response time in an $O(n^2)$ manner, which means the performance of system is degrading. The increase in response time is due to the fact that, as the number of sites increases, the overall rate of arrival of transactions into the system also increases, causing additional delays on the processing of transactions. The DVA is less sensitive to the number of nodes in the network than the CLA. Because in the CLA, when number of nodes gets larger a heavier I O load is present at the central node, but in the DVA the I O load is distributed among the sites.

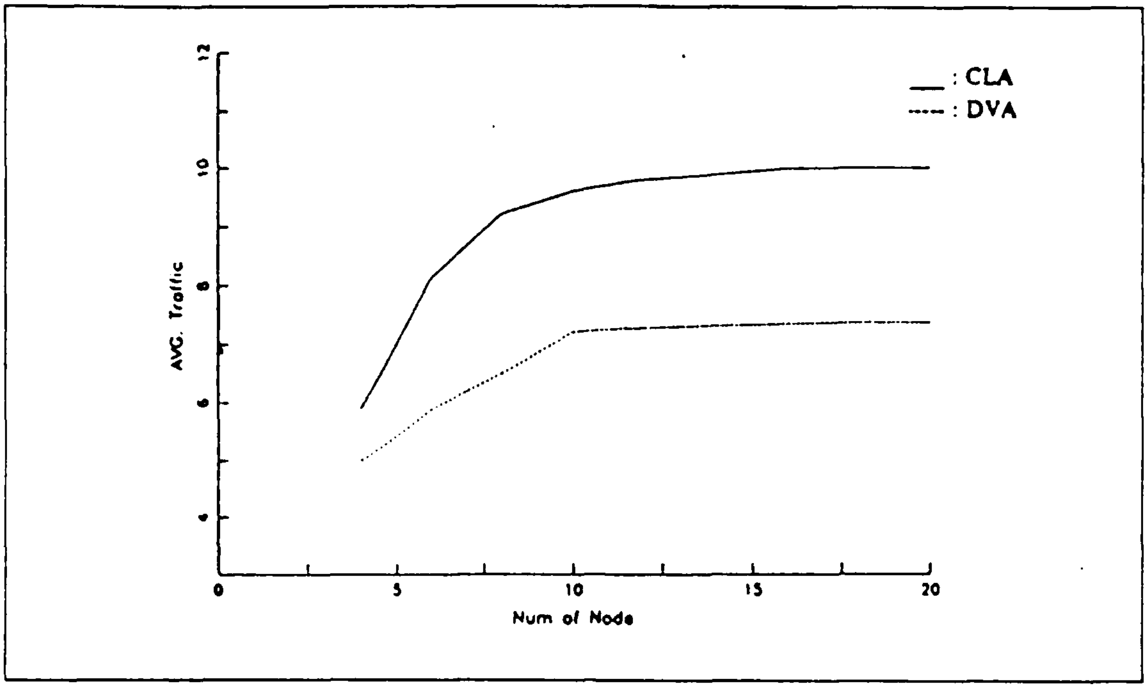

Figure 12. Effect of Number of Nodes on Average Traffic

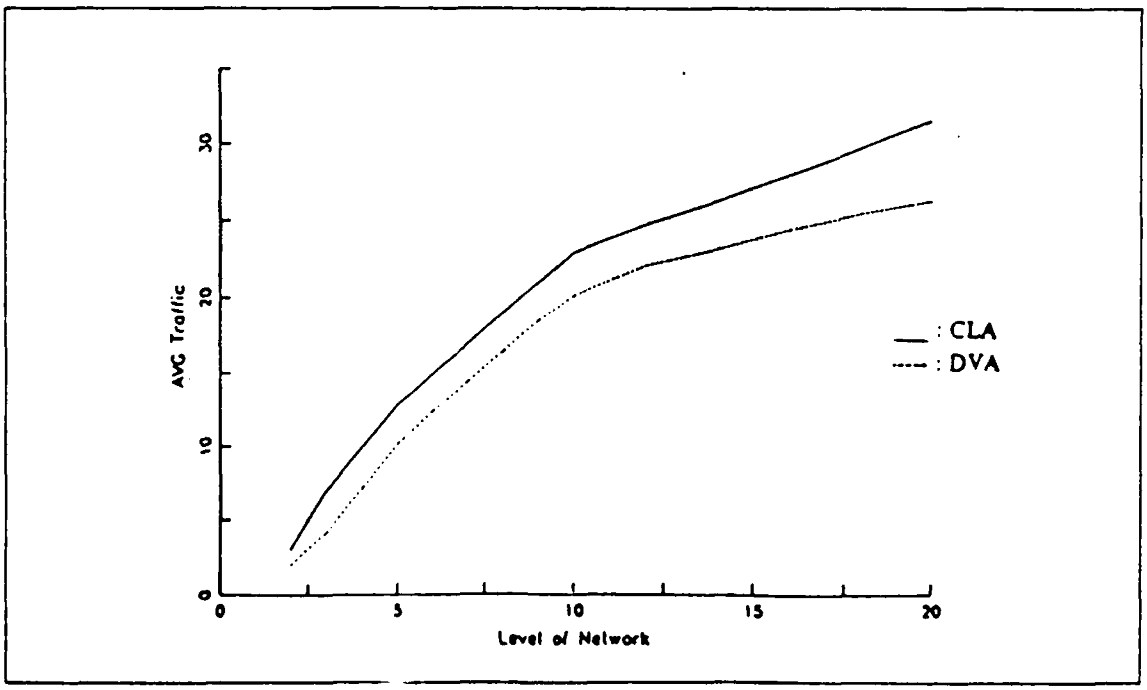

Figure 13. Effect of Level of Network on Average Traffic

The effect of the level of the network on the response time is depicted in Figure 8. This result shows that the level of hierarchical network, given the same number of nodes, has a direct impact on the response time. The impact is not entirely linear. It indicates a tendency that the performance will be degraded exponentially. This graph is somewhat complicated, so we consider 3 cases depending on the level of network, and investigate the response time versus interarrival time for each case. The results of these are shown Figure 9, Figure 10, and Figure 11. Figure 9 shows the effect of interarrival time on response time when L is small (i.e., $L < 3$). The CLA performs better than the DVA in this case. This is quite reasonable because, when L is small, the traffic of the system is not as important a factor from a performance point of view. Thus, the CLA takes less time to perform the transaction than the DVA. DVA needs extra I O time for timestamp for S[F]. Figure 10 shows case II (i.e., $3 < L < 15$). In this case, the DVA performs better than the CLA, because if L gets larger, the traffic for system becomes more important and, for conflict transaction, the transaction has to get to the central node in order to have locks in CLA. However, for DVA the transaction will be rejected before getting to the central node. This behavior saves much of the DVA's response time. Figure 11 shows an extreme case (i.e., $L > 15$). In this case the two algorithms are almost identical in performance.

Figure 12 shows the average traffic versus the number of nodes in the network. For a given level of the network, the average traffic of the system is increased sharply when number of nodes get to 10, and it will be saturated after the number of nodes exceeds 10.

Figure 13 shows the average traffic of the network versus the level of the network. Here we observe that the increasing traffic is somewhat linear, but, when L gets larger,

the rate of increase is decreased. Even though Figure 12 and Figure 13 are similar, we can say that the more critical factor for traffic of network is the level of the network.

V. CONCLUSION

We have presented a performance comparison of concurrency control algorithms. Based on the results obtained, we reached the following major conclusions:

The network structure is the most important factor for the performance of concurrency control algorithms. Especially, in a hierarchical computer network the level of network is the most critical factor to determine the performance of concurrency control algorithms.

Although we discuss each parameter individually, a network designer should note that all of them are related and should consider their combined effects on the network performance. The tradeoffs among different parameters may be affected by other operational policy that exists in the system. To have each node exhibiting various transaction rates and interarrival rates is one of the areas that needs future investigation when a network designer wants to customize or condition the network to his need.

This study was the first trial considering the hierarchical structure of a network in performance analysis of the concurrency control area. During simulation of various hierarchical computer networks we find that it is hard to describe the network structure precisely. For example, even though a network has the same number of levels and same number of nodes, there are still many possible system configurations. There is a need for a better network representation in terms of data structure and computation algorithms.

In order to maximize the performance of the network the choice of concurrency control algorithms is very important. Since the network structure may be changed dynamically so we may need to develop dynamic concurrency algorithms which can perform best under various system structures.

APPENDIX A. SIMULATION PROGRAM LIST FOR CLA

```
'' ****************************************************
'' This program is written for simulation of CLA
'' ****************************************************

''****************************************************
PREAMBLE
''****************************************************

NORMALLY, MODE IS INTEGER

DEFINE RESPONSE.TIME TO MEAN ATRB1
DEFINE AVERAGE.BASESET TO MEAN ATRB2
DEFINE COMM.TRAFFIC  TO MEAN ATRB3

EVENT NOTICES INCLUDE TRANSACTION.GEN, ARRIVAL.AT.CENTRAL,
                      RELEASE.LOCK, UPDATE
EVERY ARRIVAL.AT.CENTRAL HAS A RECEIVE.TRANSACTION
EVERY RELEASE.LOCK HAS A DONE.TRANSACTION
EVERY UPDATE HAS A UPDATE.TRANSACTION

TEMPORARY ENTITIES
EVERY TRANSACTION HAS A GEN.TIME,  A TRANSACTION.SEQ, A MESSAGE.TYPE,
           A NODE.NUM, A NODE.LEVEL, A NUM.OF.BASESET, A BASE.ITEM
EVERY TRANSACTION MAY BELONG TO THE QUEUE1,
                AND MAY BELONG TO THE WAIT.QUEUE

THE SYSTEM OWNS THE QUEUE1, THE WAIT.QUEUE
DEFINE QUEUE1 AS A FIFO SET
DEFINE WAIT.QUEUE AS A FIFO SET
DEFINE INITIATE AS A  ROUTINE
DEFINE CHECK.LOCK AS A ROUTINE
DEFINE REPORT.GENERATOR AS A ROUTINE
DEFINE TEST AS A ROUTINE
DEFINE LOCK.LIST AS AN INTEGER, 1-DIMENSIONAL ARRAY
DEFINE BASE.TABLE AS AN INTEGER, 2-DIMENSIONAL ARRAY

DEFINE GEN.TIME, RESPONSE.TIME, AVERAGE.BASESET, T, Cu, Id, Cs,
       COMM.TRAFFIC, RATE AS A REAL VARIABLES
DEFINE TRANSACTION.NUM, TRANSACTION.SEQ, MESSAGE.TYPE, NODE.NUM,
       NODE.LEVEL, NUM.OF.BASESET, QUE.INDEX, LASTNODE,
       MINBASE, MAXBASE, LOCK.STATUS, Dn, LIMITofTRANS, MAXLEVEL,
       BASE.ITEM, COUNT, SYSTEM.STATUS  AS A INTEGER VARIABLES

TALLY AVE.RESPONSE AS THE AVERAGE OF RESPONSE.TIME
TALLY AVE.TRAFFIC AS THE AVERAGE OF COMM.TRAFFIC

END
```

```
''*************************************************************
 MAIN
''*************************************************************

 RESERVE BASE.TABLE(*,*) AS 2000  BY 5

 RESERVE LOCK.LIST AS 2000
 READ SYS_RATE
 RATE = SYS_RATE/20
 SCHEDULE A TRANSACTION.GEN NOW

 LET  T = 0.08            ''  MESSAGE TRANSMISSION TIME
 LET  Cu = 0.001          '' CPU TIME SLICE TO COMPUTE ACTUAL VALUE OF TRANS
 LET  Cs = 0.00001        '' CPU TIMESLICE FOR SMALL COMPUTATION
 LET  Id = 0.025          '' I/O TIME SLICE TO READ OR WRITE FROM DATABASE
 LET  LASTNODE = 20       '' LAST NODE NUMBER
 LET  MINBASE = 1         '' MINIMUM NUMBER OF BASESET
 LET  MAXBASE = 5         '' MAXIMUM NUMBEROF BASESET
 LET  QUE.INDEX = 0       '' QUEINDEX
 LET  LOCK.STATUS = 0     '' IF LOCKED ALREADY THEN 1 ELSE 0
 LET  Dn = 1000 * LASTNODE    '' TOTAL DATA ITEM IN THE DATABASE
 LET  LIMITofTRANS =1000  '' THE NUMBER OF TRANSACTION SIMMULATED
 LET  MAXLEVEL = 3        '' LEVEL OF NETWORK
 LET  COUNT = 0           '' NUMBER OF TRANSACTION
 LET  TRANSACTION.NUM = 0
 LET SYSTEM.STATUS = 0
 START SIMULATION
 END

''*************************************************************
EVENT TRANSACTION.GEN
''*************************************************************

CREATE  A TRANSACTION
ADD 1 TO TRANSACTION.NUM
IF TRANSACTION.NUM IS GE LIMITofTRANS
   CALL REPORT.GENERATOR
ALWAYS
LET GEN.TIME = TIME.V
LET MESSAGE.TYPE = 1
CALL INITIATE
IF NODE.NUM = 1
        IF SYSTEM.STATUS = 0
           CALL CHECK.LOCK
        ELSE
           FILE TRANSACTION IN THE WAIT.QUEUE
        REGARDLESS
ELSE
        SCHEDULE AN ARRIVAL.AT.CENTRAL GIVEN TRANSACTION
        IN (T * (NODE.LEVEL - 1 )) MINUTES
REGARDLESS
SCHEDULE A TRANSACTION.GEN IN (EXPONENTIAL.F(RATE,1)) MINUTES
RETURN
END
```

```
''*************************************************************
ROUTINE INITIATE
''*************************************************************

 ADD 1 TO TRANSACTION.SEQ
 NODE.NUM = TRUNC.F(UNIFORM.F(1,LASTNODE,1))
 IF NODE.NUM = 1    NODE.LEVEL = 1
 ELSE
     IF NODE.NUM = 2 OR NODE.NUM = 3     NODE.LEVEL = 2
     ELSE
        NODE.LEVEL = 3
     REGARDLESS
 REGARDLESS
NUM.OF.BASESET = TRUNC.F(UNIFORM.F(MINBASE,MAXBASE,1))
LET J = TRANSACTION.SEQ
  FOR I = 1 TO NUM.OF.BASESET
   DO
   IF NODE.NUM = 1  BASE.TABLE(J,I) = TRUNC.F(UNIFORM.F(1,2000,1))
   ALWAYS
   IF NODE.NUM = 2 BASE.TABLE(J,I) = TRUNC.F(UNIFORM.F(1,900,1))


   ALWAYS
   IF NODE.NUM = 3  BASE.TABLE(J,I) = TRUNC.F(UNIFORM.F(901,1900,1))
   ALWAYS
   IF NODE.NUM > 3 AND  NODE.NUM < 12
      BASE.TABLE(J,I) = TRUNC.F(UNIFORM.F(1,800,1))


   ALWAYS

   IF NODE.NUM > 11  BASE.TABLE(J,I) = TRUNC.F(UNIFORM.F(901,1800,1))
   ALWAYS
   LOOP
 END

''*************************************************************
EVENT ARRIVAL.AT.CENTRAL GIVEN RECEIVE.TRANSACTION
''*************************************************************

 IF MESSAGE.TYPE(RECEIVE.TRANSACTION) = 1
                       IF SYSTEM.STATUS = 0
                               SYSTEM.STATUS = 1
                               CALL CHECK.LOCK
                       ELSE
                          FILE RECEIVE.TRANSACTION IN THE WAIT.QUEUE
                       REGARDLESS
 ELSE SCHEDULE A RELEASE.LOCK GIVEN RECEIVE.TRANSACTION
        IN ( T * (NODE.LEVEL(RECEIVE.TRANSACTION)-1)) MINUTES
 REGARDLESS
 RETURN
 END
```

```
''************************************************************
ROUTINE CHECK.LOCK
''************************************************************

 LET LOCK.STATUS = 0
 FOR I = 1 TO NUM.OF.BASESET
 DO
   IF LOCK.LIST(BASE.TABLE(TRANSACTION.SEQ,I)) = 1
      LOCK.STATUS = 1
   ALWAYS
 LOOP
 IF LOCK.STATUS = 1    FILE TRANSACTION IN THE QUEUE1
                       QUE.INDEX = QUE.INDEX + 1
 ELSE
   FOR I = 1 TO NUM.OF.BASESET
   DO
     LOCK.LIST (BASE.TABLE(TRANSACTION.SEQ,I)) = 1
   LOOP
   SCHEDULE AN UPDATE GIVEN TRANSACTION
   IN (T * (NODE.LEVEL -1)) MINUTES
 REGARDLESS

END

''************************************************************
EVENT UPDATE GIVEN UPDATE.TRANSACTION
''************************************************************

 MESSAGE.TYPE(UPDATE.TRANSACTION) = 2
 SCHEDULE AN ARRIVAL.AT.CENTRAL GIVEN UPDATE.TRANSACTION IN
  (NUM.OF.BASESET(UPDATE.TRANSACTION)
   * (Cu + 2*Id) +((NODE.LEVEL(UPDATE.TRANSACTION)-1) * T)) MINUTES

RETURN
END

''************************************************************
EVENT RELEASE.LOCK GIVEN DONE.TRANSACTION
''************************************************************

 FOR I = 1 TO NUM.OF.BASESET(DONE.TRANSACTION)
 DO
   LOCK.LIST (BASE.TABLE(TRANSACTION.SEQ(DONE.TRANSACTION),I)) = 0
 LOOP
   LOCK.STATUS = 0
 LET RESPONSE.TIME = TIME.V - GEN.TIME(DONE.TRANSACTION)
 LET COMM.TRAFFIC = NODE.LEVEL(DONE.TRANSACTION) * 3
 LET AVERAGE.BASESET = NUM.OF.BASESET
 SYSTEM.STATUS = 0

 IF QUEUE1 IS NOT EMPTY
          REMOVE THE FIRST TRANSACTION FROM THE QUEUE1
          QUE.INDEX = QUE.INDEX - 1
          SYSTEM.STATUS = 1
          CALL CHECK.LOCK
ELSE
```

```
    IF WAIT.QUEUE IS NOT EMPTY
          REMOVE THE FIRST TRANSACTION FROM THE WAIT.QUEUE
          SYSTEM.STATUS = 1
          CALL CHECK.LOCK
    ALWAYS
REGARDLESS
RETURN
END

''*************************************************************
ROUTINE REPORT.GENERATOR
''*************************************************************

 SKIP 2 OUTPUT LINES
 PRINT 1 LINE THUS
 SIMMULATION RESULT OF TWO PHASE LOCKING ALGORITHM
 SKIP 1 LINE
 PRINT 2 LINE THUS
 I. MODEL INPUT PARAMETER
    a. MODEL DESCRIPTION
 PRINT  3 LINE WITH MAXLEVEL, LASTNODE, AND Dn THUS
        1. LEVEL OF THE NETWORK:               **
        2. NUMBER OF NODE IN NETWORK:          ***
        3. DATA ITEMS IN NETWORK:             ****
 PRINT 1 LINE THUS
    b. SIMMULATION PARAMETERS
 PRINT 3 LINE WITH LIMITofTRANS, RATE, AND MAXBASE THUS
        1. NUMBER OF TRANSACTION SIMMULATED :        *****
        2. MEAN INTERARRIVAL TIME :                    *.*
        3. MAXIMUM BASESET :                             *
 PRINT 1 LINE THUS
 II. RESULTS OF SIMMULATION

 PRINT 2 LINE WITH AVE.RESPONSE*HOURS.V*MINUTES.V,
   AVE.TRAFFIC THUS
   a. THE AVERAGE RESPONSE TIME OF UPDATE TRANSACTION :   ***.*
   b. THE AVERAGE NUMBER OF TRAFFIC :                      ***
   STOP
   END
```

APPENDIX B. SIMULATION PROGRAM LIST FOR DVA

```
''***************************************************************
PREAMBLE
''***************************************************************

 NORMALLY, MODE IS INTEGER

 DEFINE RESPONSE.TIME TO MEAN ATRB1
 DEFINE AVERAGE.BASESET TO MEAN ATRB2
 DEFINE COMM.TRAFFIC  TO MEAN ATRB3

 EVENT NOTICES INCLUDE TRANSACTION.GEN, VOTING,
                       CHECK_OK
 EVERY VOTING HAS A VOTING.TRANSACTION
 EVERY CHECK_OK HAS A CHECKING.TRANSACTION


 TEMPORARY ENTITIES
 EVERY TRANSACTION HAS A GEN.TIME,  A TRANSACTION.SEQ, A MESSAGE.TYPE,
           A NODE.NUM, A NODE.LEVEL, A NUM.OF.BASESET, A BASE.ITEM,
           A TIME_STAMP, A CURRENT_NODE
 EVERY TRANSACTION MAY BELONG TO THE QUEUE1,
                AND MAY BELONG TO THE WAIT.QUEUE

 THE SYSTEM OWNS THE QUEUE1, THE WAIT.QUEUE
 DEFINE QUEUE1 AS A FIFO SET
 DEFINE WAIT.QUEUE AS A FIFO SET
 DEFINE INITIATE AS A  ROUTINE
 DEFINE INPUTRTN AS A  ROUTINE
 DEFINE FINDSET  AS A  ROUTINE
 DEFINE INITIAL_TAB AS A ROUTINE
 DEFINE SEND_ACCEPT AS A ROUTINE
 DEFINE SEND_REJECT AS A ROUTINE
 DEFINE SYSTEM_CONTROL AS A ROUTINE
 DEFINE EN_QUEUE AS A ROUTINE
 DEFINE REPORT.GENERATOR AS A ROUTINE

 DEFINE CHILDREN_TAB AS AN INTEGER, 2-DIMENSIONAL ARRAY
 DEFINE PARENT_TAB AS AN INTEGER, 1-DIMENSIONAL ARRAY
 DEFINE DATA_DIR AS AN INTEGER, 2-DIMENSIONAL ARRAY
 DEFINE LEVEL_TAB AS AN INTEGER, 1-DIMENSIONAL ARRAY
 DEFINE BASE.TABLE AS AN INTEGER, 2-DIMENSIONAL ARRAY
 DEFINE SET_TAB AS AN INTEGER, 2-DIMENSIONAL ARRAY
 DEFINE TSofDB AS A REAL, 2-DIMENSIONAL ARRAY
 DEFINE STATUS_TAB AS AN INTEGER, 1-DIMENSIONAL ARRAY


 DEFINE GEN.TIME, RESPONSE.TIME, AVERAGE.BASESET, T, Cu, Id, Cs,
        COMM.TRAFFIC, TIME_STAMP,RATE AS A REAL VARIABLES
 DEFINE TRANSACTION.NUM, TRANSACTION.SEQ, MESSAGE.TYPE, NODE.NUM,
```

```
       NODE.LEVEL, NUM.OF.BASESET, QUE.INDEX, LASTNODE,CURRENT_NODE,
       MINBASE, MAXBASE, LOCK.STATUS, Dn, LIMITofTRANS, MAXLEVEL,
       BASE.ITEM, COUNT, SYSTEM.STATUS, YESNO AS A INTEGER VARIABLES
 DEFINE MAX_PATH AS AN INTEGER VARIABLE

 TALLY AVE.RESPONSE AS THE AVERAGE OF RESPONSE.TIME
 TALLY AVE.BASESET AS THE AVERAGE OF AVERAGE.BASESET
 TALLY AVE.TRAFFIC AS THE AVERAGE OF COMM.TRAFFIC

END


''***************************************************
MAIN
''***************************************************

CALL INITIAL_TAB
PRINT 1 LINE THUS
DO YOU WANT DEFAULT MODEL ? IF YES TYPE 1
READ YESNO

IF YESNO <> 1
CALL INPUT_RTN
ALWAYS

 SCHEDULE A TRANSACTION.GEN NOW

 LET  T = 0.2            '' MESSAGE TRANSMISSION TIME
 LET  Cu = 0.001         '' CPU TIME SLICE TO COMPUTE ACTUAL VALUE OF TRANS
 LET  Cs = 0.00001       '' CPU TIMESLICE FOR SMALL COMPUTATION
 LET  Id = 0.025         '' I/O TIME SLICE TO READ OR WRITE FROM DATABASE
 LET  LASTNODE = 10      '' LAST NODE NUMBER
 LET  MINBASE = 1        '' MINIMUM NUMBER OF BASESET
 LET  MAXBASE = 5        '' MAXIMUM NUMBEROF BASESET
 LET  QUE.INDEX = 0      '' QUEINDEX
 LET  LOCK.STATUS = 0    '' IF LOCKED ALREADY THEN 1 ELSE 0
 LET  Dn = 1000          '' TOTAL DATA ITEM IN THE DATABASE
 LET  LIMITofTRANS =1000   '' THE NUMBER OF TRANSACTION SIMMULATED
 LET  MAXLEVEL = 3       '' LEVEL OF NETWORK
 LET  COUNT = 0          '' NUMBER OF TRANSACTION
 LET  TRANSACTION.NUM = 0
 LET SYSTEM.STATUS = 0
 LET RATE = 3
 START SIMULATION
 END


''***************************************************
EVENT TRANSACTION.GEN
''***************************************************

CREATE  A TRANSACTION
ADD 1 TO TRANSACTION.NUM
LET TRANSACTION.SEQ = TRANSACTION.NUM
IF TRANSACTION.NUM IS GE LIMITofTRANS
```

```
   CALL REPORT.GENERATOR
ALWAYS
LET GEN.TIME = TIME.V
LET TIME_STAMP = TIME.V
CALL INITIATE
CALL FIND_SET


SCHEDULE A TRANSACTION.GEN IN (EXPONENTIAL.F(RATE,1)) MINUTES
RETURN

END


''**************************************************
ROUTINE  INITIAL_TAB
''**************************************************

RESERVE CHILDREN_TAB(*,*) AS 30 BY 30
RESERVE PARENT_TAB, LEVEL_TAB, STATUS_TAB AS 30
RESERVE DATA_DIR(*,*) AS 30 BY 2
RESERVE BASE.TABLE(*,*) AS 1000 BY 5
RESERVE SET_TAB(*,*) AS 5 BY 30
RESERVE TSofDB(*,*) AS 30 BY 1000

FOR I = 1 TO 1000
DO
    FOR J = 1 TO 5
    DO
        BASE.TABLE(I,J) = 0
    LOOP
LOOP
FOR  I = 1 TO 30
DO
   PARENT_TAB(I) = 0
   LEVEL_TAB(I) = 0
   STATUS_TAB(I) = 1
   FOR J = 1 TO 30
   DO
       CHILDREN_TAB(I,J) = 0
   LOOP
   FOR J = 1 TO 2
   DO
       DATA_DIR(I,J) = 0
   LOOP
   FOR J = 1 TO 1000
   DO
       TSofDB(I,J) = 0.0
   LOOP
LOOP
FOR I = 1 TO 5
DO
    FOR J = 1 TO 30
    DO
        SET_TAB(I,J) = 0
```

```
    LOOP
LOOP
RELEASE CHILDREN_TAB, PARENT_TAB, DATA_DIR, LEVEL_TAB, BASE.TABLE,
        SET_TAB, TSoFDB, STATUS_TAB

END


''**************************************************************
ROUTINE INITIATE
''**************************************************************

 RESERVE LEVEL_TAB AS  30
 RESERVE DATA_DIR(*,*) AS 30 BY 2
 RESERVE BASE.TABLE(*,*) AS 1000 BY 5
 ADD 1 TO TRANSACTION.SEQ
 NODE.NUM = TRUNC.F(UNIFORM.F(1,LASTNODE,1))
 LET CURRENT_NODE = NODE.NUM

 IF YESNO = 1      '' ******** IF DEFAULT NETWORK  ******
    IF NODE.NUM = 1    NODE.LEVEL = 1
    ELSE
     IF NODE.NUM = 2 OR NODE.NUM = 3 OR NODE.NUM = 4    NODE.LEVEL = 2
     ELSE
        NODE.LEVEL = 3
     REGARDLESS
    REGARDLESS


NUM.OF.BASESET = TRUNC.F(UNIFORM.F(MINBASE,MAXBASE,1))
LET J = TRANSACTION.SEQ
  FOR I = 1 TO NUM.OF.BASESET
   DO
   IF NODE.NUM = 1  BASE.TABLE(J,I) = TRUNC.F(UNIFORM.F(001,1000,1))
                    DATA_DIR(1,1) = 001
                    DATA_DIR(1,2) = 1000
   ALWAYS
   IF NODE.NUM = 2  BASE.TABLE(J,I) = TRUNC.F(UNIFORM.F(001,400,1))
                    DATA_DIR(2,1) = 001
                    DATA_DIR(2,2) = 400
   ALWAYS
   IF NODE.NUM = 3  BASE.TABLE(J,I) = TRUNC.F(UNIFORM.F(401,700,1))
                    DATA_DIR(3,1) = 401
                    DATA_DIR(3,1) = 700
   ALWAYS
   IF NODE.NUM = 4  BASE.TABLE(J,I) = TRUNC.F(UNIFORM.F(701,900,1))
                    DATA_DIR(4,1) = 701
                    DATA_DIR(4,2) = 900
   ALWAYS
   IF NODE.NUM = 5  BASE.TABLE(J,I) = TRUNC.F(UNIFORM.F(1,100,1))
                    DATA_DIR(5,1) = 001
                    DATA_DIR(5,2) = 100
   ALWAYS
   IF NODE.NUM = 6  BASE.TABLE(J,I) = TRUNC.F(UNIFORM.F(101,200,1))
                    DATA_DIR(6,1) = 101
                    DATA_DIR(6,2) = 200
```

```
    ALWAYS
    IF NODE.NUM = 7   BASE.TABLE(J,I) = TRUNC.F(UNIFORM.F(201,300,1))
                      DATA_DIR(7,1) = 201
                      DATA_DIR(7,2) = 300
    ALWAYS
    IF NODE.NUM = 8   BASE.TABLE(J,I) = TRUNC.F(UNIFORM.F(401,500,1))
                      DATA_DIR(8,1) = 401
                      DATA_DIR(8,2) = 500
    ALWAYS
    IF NODE.NUM = 9   BASE.TABLE(J,I) = TRUNC.F(UNIFORM.F(501,600,1))
                      DATA_DIR(9,1) = 501
                      DATA_DIR(9,2) = 600
    ALWAYS
    IF NODE.NUM = 10  BASE.TABLE(J,I) = TRUNC.F(UNIFORM.F(701,800,1))
                      DATA_DIR(10,1) = 701
                      DATA_DIR(10,2) = 800
    ALWAYS
    LOOP
 LET LEVEL_TAB(1) = 1
 FOR I = 2 TO 4
 DO
 LET LEVEL_TAB(I) = 2
 LOOP
 FOR I = 5 TO 10
 DO
 LET LEVEL_TAB(I) = 3
 LOOP
 ELSE  '' ******** IF NOT DEFAULT NETWORK ********************
  NODE.LEVEL = LEVEL_TAB(NODE.NUM)
  NUM.OF.BASESET = TRUNC.F(UNIFORM.F(MINBASE,MAXBASE,1))
  LET J = TRANSACTION.NUM
  FOR I = 1 TO NUM.OF.BASESET
  DO
    BASE.TABLE(J,I) = TRUNC.F(UNIFORM.F(DATA_DIR(NODE.NUM,1),
                                         DATA_DIR(NODE.NUM,2),1))
  LOOP
  REGARDLESS

 END


 ''***************************************************
 ROUTINE INPUT_RTN
 ''***************************************************


 RESERVE CHILDREN_TAB(*,*) AS 30 BY 30
 RESERVE LEVEL_TAB  AS 30
 RESERVE DATA_DIR(*,*) AS 30 BY 2
 RESERVE PARENT_TAB AS 30
 LET INDX = 2
 PRINT 2 LINE THUS
 INPUT PARAMETERS CAREFULLY !!
 HOW MANY NODES THE NETWORK HAVE ? '
 READ LASTNODE
 PRINT 1 LINE THUS
```

```
WHAT IS THE LEVEL OF THE NETWORK ?
READ MAXLEVEL
FOR I = 1 TO LASTNODE
DO
  PRINT 1 LINE WITH I THUS
  HOW MANY CHILDREN DODES NODE ** HAVE ?
  READ J
   IF J <> 0
     FOR K = 1 TO J
     DO
       IF INDX <= LASTNODE
         CHILDREN_TAB(I,K) = INDX
         PARENT_TAB(INDX) = I
         INDX = INDX + 1
       ALWAYS
     LOOP
   ALWAYS
LOOP

FOR  J = 1 TO LASTNODE
DO
PRINT 1 LINE  WITH J THUS
WHAT IS THE LEVEL OF NODE ** AND RANGE OF DATA ITEM ?
READ LEVEL_TAB(J), DATA_DIR(J,1), AND DATA_DIR(J,2)
LOOP

PRINT 1 LINE THUS
THANK YOUR INPUT IS DONE !!!

END

''*************************************************************
ROUTINE FIND_SET
''*************************************************************



'' THIS ROUTINE GENERATE SETS WHICH S°i  AND S°F
RESERVE LEVEL_TAB AS 30
RESERVE SET_TAB(*,*) AS 5 BY 30
LET MAX_PATH = 0
FOR I = 1  TO  NUM.OF.BASESET
DO
    FOR  J = 1 TO LASTNODE
    DO
      IF BASE.TABLE(TRANSACTION.SEQ,I) >= DATA_DIR(J,1)  AND
         BASE.TABLE(TRANSACTION.SEQ,I) <= DATA_DIR(J,2)
         SET_TAB(I,J) = 1
      ALWAYS
    LOOP
LOOP

FOR I = 1 TO LASTNODE
DO
LET BASE_COUNT = 0
LET NODE_STATUS = 0
```

```
 FOR J = 1 TO NUM.OF.BASESET
 DO
    IF SET_TAB(J,I) <> 0
       BASE_COUNT = BASE_COUNT + 1
       NODE_STATUS = 1
    ALWAYS
 LOOP
IF NODE_STATUS = 1
   LET CURRENT_NODE = I
   IF MAX_PATH <= ABS.F(LEVEL_TAB(NODE.NUM) - LEVEL_TAB(CURRENT_NODE))
      MAX_PATH = ABS.F(LEVEL_TAB(NODE.NUM) - LEVEL_TAB(CURRENT_NODE))
   ALWAYS
PRINT 5 LINE WITH NODE.NUM, CURRENT_NODE, LEVEL_TAB(NODE.NUM),
                  LEVEL_TAB(CURRENT_NODE), BASE_COUNT THUS
    NODE.NUM =    **
  CURRENT_NODE = **
  LEVEL OF NODE.NUM = **
  LEVEL OF CURRENT NODE = **
  BASE COUNT = **
   SCHEDULE A VOTING GIVEN TRANSACTION IN
     ( (ABS.F(LEVEL_TAB(NODE.NUM) - LEVEL_TAB(CURRENT_NODE)) * T) +
       (BASE_COUNT * Id )) MINUTES
ALWAYS
LOOP
SCHEDULE A CHECK_OK GIVEN TRANSACTION IN ((MAX_PATH * T) + NUM.OF.BASESET)
         MINUTES

END


''*************************************************************
EVENT VOTING GIVEN VOTING.TRANSACTION
''*************************************************************


RESERVE TSofDB(*,*) AS  30 BY 1000
RESERVE STATUS_TAB AS 30
LET OK_STATUS = 0
FOR K = 1 TO NUM.OF.BASESET(VOTING.TRANSACTION)
DO
IF SET_TAB(K,CURRENT_NODE(VOTING.TRANSACTION)) <> 0
 IF TIME_STAMP(VOTING.TRANSACTION) >= TSofDB(CURRENT_NODE(VOTING.TRANSACTION),
                 BASE.TABLE(TRANSACTION.SEQ(VOTING.TRANSACTION),K)
    OK_STATUS = 1
 ALWAYS
ALWAYS
LOOP
IF OK_STATUS = 1
   TIME_STAMP(VOTING.TRANSACTION) = TIME.V
   STATUS_TAB(CURRENT_NODE(VOTING.TRANSACTION)) = 1
ELSE
   STATUS_TAB(CURRENT_NODE(VOTING.TRANSACTION)) =0
REGARDLESS
RETURN

END
```

```
''*************************************************************
ROUTINE SEND_ACCEPT
''*************************************************************
FOR I = 1 TO LASTNODE
DO
  FOR J = 1 TO NUM.OF.BASESET(TRANSACTION)
  DO
    IF SET_TAB(J,I) = 1
       IF TSofDB(I,J) <= TIME_STAMP(TRANSACTION)
          TSofDB(I,J) = TIME_STAMP(TRANSACTION)
       ALWAYS
    ALWAYS
  LOOP
LOOP
LET RESPONSE.TIME = TIME.V - GEN.TIME(TRANSACTION)
FOR I = 1 TO LASTNODE
DO
 FOR J = 1 TO NUM.OF.BASESET(TRANSACTION)
 DO
     SET_TAB(J,I) = 0
 LOOP
 STATUS_TAB(I) = 0
LOOP
CALL SYSTEM_CONTROL

END

''*************************************************************
ROUTINE SEND_REJECT
''*************************************************************


CALL EN_QUEUE
CALL SYSTEM_CONTROL
END

***************************************************************
ROUTINE EN_QUEUE
FILE TRANSACTION IN THE QUEUE1
''ALWAYS
'' ****  KEEP CONTINUE
END
ROUTINE SYSTEM_CONTROL
IF QUEUE1 IS NOT EMPTY
   REMOVE FIRST TRANSACTION FROM THE QUEUE1
   CALL FIND_SET
ELSE

   SCHEDULE TRANSACTION.GEN NOW
ALWAYS
END
''*************************************************************
```

```
''*************************************************************
EVENT CHECK_OK GIVEN CHECKING.TRANSACTION
''*************************************************************

RESERVE STATUS_TAB AS 30
ITS_OK = 1
FOR I = 1 TO LASTNODE
DO
ITS_OK = ITS_OK * STATUS_TAB(I)
LOOP
IF ITS_OK <> 1
   TIME_STAMP(CHECKING.TRANSACTION) = TIME.V
   CALL SEND_ACCEPT
ELSE
   CALL SEND_REJECT
REGARDLESS
RETURN
END

''*************************************************************
ROUTINE REPORT.GENERATOR
''*************************************************************


 SKIP 2 OUTPUT LINES
 PRINT 1 LINE THUS
 SIMMULATION RESULT OF TWO PHASE LOCKING ALGORITHM
 SKIP 1 LINE
 PRINT 2 LINE THUS
 I. MODEL INPUT PARAMETER
    a. MODEL DESCRIPTION
 PRINT  3 LINE WITH MAXLEVEL, LASTNODE, AND Dn THUS
       1. LEVEL OF THE NETWORK:                **
       2. NUMBER OF NODE IN NETWORK:          ***
       3. DATA ITEMS IN NETWORK:             ****
 PRINT 1 LINE THUS
    b. SIMMULATION PARAMETERS
 PRINT 3 LINE WITH LIMITofTRANS, RATE, AND MAXBASE THUS
       1. NUMBER OF TRANSACTION SIMMULATED :       ****
       2. MEAN INTERARRIVAL TIME :                  *.*
       3. MAXIMUM BASESET :                           *
 PRINT 1 LINE THUS
 II. RESULTS OF SIMMULATION

 PRINT 2 LINE WITH AVE.RESPONSE*HOURS.V*MINUTES.V, MINBASE THUS
   a. THE AVERAGE RESPONSE TIME OF UPDATE TRANSACTION :  **.*
   b. THE AVERAGE NUMBER OF BASESET:                     **
   STOP
   END
```

LIST OF REFERENCES

1. Hector Garcia-Molina, *Performance of update algorithms for replicated data,* UMI Research Press, 1981.

2. Alan Demer, Dan Greene, and Carl Hauser, *Epidemic algorithms for replicated database maintenance,* Proceeding of 6th Annual ACM symposium on principles of distributed computing, pp. 1-11, 1987.

3. W. K. Cheng, *Performance analysis of update synchronization algorithms for distributed database,* Ph.D Thesis, Department of Computer Science, University of Illinois at Urbana-Champain, Urbana, Illinois, 1981.

4. Tse-Men Koon, and M. Tamer Ozsu, *Performance comparison of resilient concurrency control algorithms of distributed databases,* International Conference on Data Engineering, pp. 565-573, 1986.

5. Gerorge A. Champine, Ronald D. Coop, and Russel C. Heinselman, *Distributed computer systems impact on management, design, and analysis,* North-Holland Publishing Co. 1980.

6. C. P. Wang and Victor O. K. Li, G. L., *A unified concurrency control algorithms for distributed database system,* 4th International Conference on Data Engineering, pp. 410-417, 1988.

INITIAL DISTRIBUTION LIST

		No. Copies
1.	Defense Technical Information Center Cameron Station Alexandria, VA 22304-6145	2
2.	Library, Code 0142 Naval Postgraduate School Monterey, CA 93943-5002	2
3.	Department Chairman, Code EC Department of Electrical and Computer Engineering Naval Postgraduate School Monterey, CA 93943-5000	1
4.	Professor Chyan Yang, Code EC Ya Department of Electrical and Computer Engineering Naval Postgraduate School Monterey, CA 93943-5000	6
5.	Professor Myung W. Suh Code AS Su Department of Adminstrative Science Naval Postgraduate School Monterey, CA 93943-5000	2
6.	ROK Army Headquarters Nonsan-Gun, Duma-Myen, Bunam-ri, CPO Box #2 Chungchungnam-Do, 320-919 Republic of Korea	2
7.	Seo, Yong Seok SMC #1448 Naval Postgraduate School Monterey, Ca 93943	1
8.	Song, Il Yeon SMC #2302 Naval Postgraduate School Monterey, Ca 93943	1
9.	Jung, Jae Doo SMC #1504 Naval Postgraduate School Monterey, Ca 93943	1

10.	Hwang, Jung Sub SMC #1209 Naval Postgraduate School Monterey, Ca 93943	1
11.	Choi, Byung Gook SMC #1587 Naval Postgraduate School Monterey, Ca 93943	1
12.	Ryoo, Moo Bong Chungchungbuk-Do, Chungwon-Gun, Sancheok-Myen, Youngdeok-Ri Hayoung Republic of Korea	1
13.	Shin, in sub Kayagok-Myun, Sannori 518 Nonsan-Gun, Chung Nam 320-840 Republic of Korea	1
13.	Shin, Eon Seok Seoul, Mapo-Gu, Mangwon-2Dong, Seokyo APT A-Dong, 303Ho Republic of Korea	6